A Comparative Study of Control Architectures in UAV/UGV-based Surveillance System

Amirreza M. Khaleghi, Dong Xu, Sara Minaeian, Mingyang Li, Yifei Yuan, Jian Liu, Young-Jun Son

Systems and Industrial Engineering, The University of Arizona, Tucson, AZ 85721, USA

Christopher Vo, Arsalan Mousavian, Jyh-Ming Lien

Computer Science, George Mason University, Fairfax, VA 22030, USA

Abstract

The goal of this paper is to study the characteristics of various control architectures (e.g. centralized, hierarchical, distributed, and hybrid) for a team of unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) in performing collaborative surveillance and crowd control. To this end, an overview of different control architectures is first provided covering their functionalities and interactions. Then, three major functional modules needed for crowd control are discussed under those architectures, including 1) crowd detection using computer vision algorithms, 2) crowd tracking using an enhanced information aggregation strategy, and 3) vehicles motion planning using a graph search algorithm. Depending on the architectures, these modules can be placed in the ground control center or embedded in each vehicle. To test and demonstrate characteristics of various control architectures, a test-bed has been developed involving these modules and various hardware and software components, such as 1) assembled UAVs and UGV, 2) a real-time simulator (in Repast Simphony), 3) off-the-shelf ARM architecture computers (ODROID-U2/3), 4) autopilot units with GPS sensors, and 5) multipoint wireless networks using XBee. Experiments successfully demonstrate the pros and cons of the considered control architectures in terms of computational performance in responding to different system conditions (e.g. information sharing).

Keywords
Control architecture, UAV, UGV, crowd control, information aggregation

1. Introduction and Background

Surveillance plays a vital role in detecting intrusions in border patrol. In recent years, the roles of stationary surveillance equipment (e.g. ground sensor, light tower, and remote video surveillance systems) have been complimented by mobile surveillance equipment such as unmanned aerial vehicle (UAV) and unmanned ground vehicle (UGV). It was pointed out in [1] that the supplemental appropriations Bill of FY2010 have included $32 million for the new UAVs’ acquisition by US Customs and Border Protection (CBP) at the Department of Homeland Security (DHS). Furthermore, by 2010, the usage of unmanned vehicles has helped to seize more than 22,000 pounds of marijuana and 5,000 illegal immigrants. Its importance grows tremendously as Federal Aviation Administration (FAA) has granted a certificate of authorization to CBP for UAV flight along the Texas border and Gulf region [1].

Surveillance and crowd control using a team of UAVs and UGVs (the theme of this paper), however, is not a trivial task. Autonomous operation of such a complex multi-vehicle system requires algorithmic design and implementation for making decisions under various constraints. Detection of the crowd, tracking their movements, and determining an optimal path for unmanned vehicles are the key decisions to consider. Essentially, these decisions can be made according to different interaction mechanisms and system structures – known as a control architecture. The system performance, computational and communicational latency can vary significantly depending on the employed control architecture. Thus, understanding the pros and cons of different architectures is a key issue in controlling a UAV/UGV system. In the field of multi-robot systems, centralized and decentralized control architectures are the most fundamental decisional classification for multi-vehicles, where decentralized architecture can furthermore be divided into hierarchical and distributed control architectures [2]. Moreover, hybrid control
architectures can be defined as a combination of different architectures. In this work, we focus on two widely adopted architectures (i.e. centralized and distributed).

In a centralized control architecture, a ground station is used as the control center processing sensory data (e.g. video streams) received from the unmanned vehicles and making various decisions (e.g. detection, tracking, motion planning). Then, control commands such as generated trajectories (i.e. sequences of waypoints) are sent back to each vehicle for execution. Later at the next decision time point, the sensory data are transmitted to the central control station again and the entire processes continue until a mission is completed. Although no major decision is made by the vehicle under this architecture, UAVs/UGVs have the autonomy for making hardware related decisions (e.g. stabilization) using the installed flight microcontroller. A benefit with the centralized control architecture is the ease of optimizing the performance from the system perspective; however, as all the sensory data is transmitted to the control center, the data transmission overhead, decision making frequency, and system scalability are the major concerns. In our previous work, we have employed this control architecture and proposed a dynamic data driven adaptive multi-scale simulation (DDDAMS) based planning and control framework for efficient decision making in UAV/UGV surveillance and crowd control [3]. In a distributed control architecture, vehicles are equipped with additional onboard data processing and decision-making units for planning and control by themselves without a control center involved. During the system operation, each unmanned vehicle collects the sensory data and makes decisions (e.g. detection, tracking, and motion planning) onboard. A coordination scheme may be needed to accomplish certain tasks, and the generated commands are directly executed onboard (no data transmission to the control center is needed). However, the onboard computational capabilities embedded in distributed architecture are not as powerful as those applied in centralized one. This is because of the different families of computer processors (x86 vs. ARM) between the ground station and onboard unit. x86 processors (usually used in a desktop computer) are known to consume lots of power, which is not suitable for the UAV/UGV powered by lightweight lithium polymer batteries. On the other hand, ARM processors consume less power and have shown promises in embedded systems such as smartphones and tablets, which make them better candidates for UAV/UGV. Major benefits of employing a distributed control architecture are the low sensory data transmission overhead and capability of deploying the entire system in a scalable manner; however, the optimality of overall system performance may be difficult to achieve, and a coordination mechanism (e.g. cooperative vs. non-cooperative) needs to be clearly defined and carefully implemented.

Given these constraints, this work aims to comparatively analyze the design, implementation and performance of major functional algorithms (i.e. crowd detection, tracking, motion planning) needed for surveillance and crowd control under different control architectures. Several multi-vehicle control architectures have been developed in literature [4-8]. However, a systematic study to compare their pros and cons is needed. In general, ease of coordination was noted as a major advantage of the centralized control architecture, and better scalability was observed under distributed control architecture in several research works. However, quantitative comparison is desired to better understand the exact boundary conditions favoring different architectures. In crowd detection, the optical flow-based algorithm is employed with various parameter customizations involved. In crowd tracking, a modified information aggregation strategy is considered to realize both tracking effectiveness and computational efficiency under different control architectures. In motion planning, the same graph search algorithm is placed either on board or on the central controller depending on the considered control architecture, and localization approach is used to improve the GPS accuracy. In addition, a significant amount of efforts has been devoted to construct an integrated testbed involving both hardware (e.g. UAV/UGV) and software (agent-based and physics-based simulations) components for configuring different control architectures.

The rest of this paper is organized as follows: Section 2 discusses the design, implementation, and performance analysis of crowd detection, crowd tracking and motion planning algorithms under different control architectures. Section 3 describes the developed control testbed for integrating various hardware and software components. Section 4 provides comparative experimental results, and Section 5 discusses the conclusion and future works.

2. Surveillance and Crowd Control Algorithms Under Different Control Architectures

2.1 Crowd Detection

The first functionality of every autonomous visual surveillance system is target detection. There are several approaches applied in the literature for detecting and classifying a target, including motion based, shape based, and color based methods. In this work, we considered motion detection for the perception of crowd location. For this
purpose, computer vision techniques have been used to process captured image sequences and videos from the environment to produce numerical or symbolic information in forms of decisions [9]. There are different algorithms (e.g. background subtraction) available in the literature for segmenting the motion from a static camera. However, in case of a moving camera, the problem is more challenging and the compensation for background motion is needed. According to [10], optical flow is the most popular analysis technique for motion detection from the aerial vehicles. Optical flow is defined as the pattern of apparent motion of objects in a visual scene caused by the relative motion between an observer and the scene [11]. Therefore, this algorithm is capable of compensating the background motion through computing the relative movements.

The optical flow-based algorithm that was presented in [12] is considered in this paper. The first step in motion detection using optical flow is to extract the features and match them through subsequent frames of the video. For this purpose, the eigenvalues of all pixels are firstly computed and ranked, and then those with high eigenvalues are picked as good features using cvGoodFeaturesToTrack function of the OpenCV (Open Source Computer Vision Library). This technique was first introduced by Shi and Tomasi [13] and implies that pixels with bigger eigenvalues have higher probability to represent texture and/or corners. Then, in order to match the related features in the subsequent frames, the motion parameter is calculated by the optical flow, based on the Pyramidal Lucas-Kanade feature tracker technique. This technique assumes the brightness constancy as well as spatio-temporal constancy and so the optical flow velocity of a moving pixel is estimated by the ratio of the time derivative of the intensity over the space derivative of the intensity [14], which is outlined as

$$V = \frac{dx}{dt} = -\frac{I_t}{I_x}$$

where, $I_x$ is the spatial derivative across the first image, and $I_t$ is the derivative between images over time. Then, a modified threshold is applied to the warped images to discriminate the foreground crowd from the background. In this work, we modified some of the parameters and selecting rules under different control architectures, including:

- Max and min time delta as the tracking parameters used in computing motion gradient,
- Number of cyclic frame buffer used for motion detection,
- The threshold to reject very small movements,
- The parameter for feature selection.

![Figure 1](image1)

(a) Detection at frame 180  
(b) Detection at frame 236  
(c) Detection at frame 310

Figure 1: Performance of the detection algorithm - graphical outputs of the program are shown in red

Figure 1 illustrates few snapshots of testing the algorithm for crowd detection, using a video from a moving camera, mounted on a UAV. As shown in the figure, some detection errors may occur in certain frames, which will be corrected automatically by following the trend of target motion in subsequent frames. Hence, the overall performance is good enough to track the real targets over a time period. The algorithm works similarly for detection from a UGV, however the video resolution differs, and hence, the parameters need to be modified accordingly.
2.2 Crowd Tracking

In the crowd tracking module, UAVs and UGVs play complementary roles in crowd information gathering. UAVs flying at a high altitude have a wider view of the crowd, but capture low-resolution data containing information such as shape and location of the whole crowd. To capture the detailed information of individuals in the crowd, UGVs can be deployed close to the crowd so that individuals could be clearly observed and detected. However, UGVs may not observe all the individuals in the crowd due to their limited detection range. An exemplary scenario from a top-down view is illustrated in Figure 2(a), where every UGV can only detect a small portion of the crowd. It is desirable to utilize both advantages of UAVs and UGVs in the crowd tracking. Information-aggregation-based tracking algorithm is proposed to achieve this objective [15]. Essentially, the algorithm attempts to provide the crowd prediction at any time t based on historical information observed from both UAVs and UGVs, which can be summarized as:

- Step 1: Crowd dynamics modeling by UAVs,
- Step 2: Individuals dynamics modeling by UGVs,
- Step 3: Crowd prediction of UAVs by aggregating UGVs’ prediction results.

Steps 1-3 are conducted iteratively at each time stamp to provide the corresponding crowd prediction. Specifically, in Step 1, crowd locations are represented by two-dimensional occupancy grid with equally-sized cells (see Figure 2(b)) due to low-resolution data observed by UAVs. Then crowd prediction can be transformed into estimating whether any cell, denoted as \(C(l,m)\), is occupied (\(C(l,m)=1\)) or not (\(C(l,m)=0\)), where \(C(l,m)\) refers to the cell at row \(l\) and column \(m\). \(p(l,m)\) quantifies the probability of \(C(l,m)\) being occupied, i.e., \(p(l,m)=Pr(C(l,m)=1)\) and is estimated under the Bayesian framework. Bayesian estimation is employed due to its advantages of integrating prior knowledge and available data. Beta prior is assigned to \(p(l,m)\) and historical data are modeled with a binomial distribution. Due to the natural conjugate relationship of beta-binomial, the estimated \(p(l,m)\) can be achieved as the mean of the beta posterior distribution as

\[
\hat{p}_A(l,m) = \frac{Z(l,m)+\alpha(l,m)}{N(l,m)+\alpha(l,m)+\beta(l,m)}
\]

where \(\hat{p}_A(l,m)\) is the estimated probability of \(C(l,m)\) being occupied at \(t+1\) based on UAVs’ data. \(\alpha(l,m)\) and \(\beta(l,m)\) are the shape parameters of beta prior distribution, which can be elicited based on prior knowledge. If no prior knowledge is available, non-informative priors, such as uniform prior of Beta(1,1) or Jeffreys’ prior of Beta(0.5,0.5), can be assigned. \(N(l,m,t)\) is the total number of historical observations collected by UAVs. \(Z(l,m,t)\) is the number of observations that \(C(l,m)\) is occupied.

In Step 2, autoregressive (AR) model is adopted to represent the individual dynamics for UGVs. Compared with other modeling approaches, AR model is suitable for scenarios where individuals’ locations are directly observed with ignorable errors. Under this condition, the model further assumes linear relationship between the individual’s location at next time stamp and the current one. Denoting the state of the \(i^{th}\) individual detected by UGVs at time \(t\) as a vector \(X_i(t)=[x_i(t), y_i(t), v_{x,i}(t), v_{y,i}(t)]^T\), where \(x_i(t)\) and \(y_i(t)\) represent the 2D location of that individual and \(v_{x,i}(t)\) and \(v_{y,i}(t)\) are its speed in \(x\) and \(y\) direction, respectively. A lag-1 AR model can be specified as

\[
X_i(t) = q_i(t-1)X_i(t-1) + B_f(t)f_i(t) + B_u(t)u_i(t) + \epsilon_i(t)
\]

where, \(q_i(t-1)\) is the time-dependent autoregressive parameter, representing individual’s changing dynamics (e.g. acceleration/deceleration, making turns). Vectors \(f_i(t)\) and \(u_i(t)\) represent the exogenous inputs from environment and UGV/UAVs, respectively. Matrices \(B_f(t)\) and \(B_u(t)\) correspond to the time-dependent effects of these inputs on the individual’s dynamics. \(\epsilon_i(t)\) is the regression error term.
With the prediction results from UAVs and UGVs, in Step 3, UAVs’ prediction with aggregated information from UGVs can be given by

$$\hat{P}(l,m) = w_e(l,m)\hat{P}_e(l,m) + (1 - w_e(l,m))\hat{P}_g(l,m)$$ (4)

where, \(\hat{P}(l,m)\) is the system estimated probability of \(C(l,m)\) being occupied at \(t+1\), \(\hat{P}_e(l,m)\) and \(\hat{P}_g(l,m)\) are the estimated probability by UAVs’ and UGVs’ data respectively; and \(w_e(l,m)\) is the weight to balance the influence of UAVs’ and UGVs’ data in the crowd prediction.

### 2.3 Motion Planning

Motion planning plays a vital role in UAV/UGV surveillance and crowd control. The goal is to select the optimal path among all the feasible solutions subject to a combination of criteria (e.g. shortest path, minimum energy consumption). In this work, we consider the static, time-invariant and differentially-unconstrained point vehicle representation (e.g. UAV/UGV) for the motion planning problem. A full list of various other types of motion planning models can be found in [16]. Considering the centralized control architecture, the ground station works as the supervisor, in which the motion planning is performed and the generated paths are transmitted to each UAV/UGV. All the decisions including multiple UAV/UGV coordination, task allocation and scheduling are planned in the centralized station. As all the sensory information (e.g. image, video) needs to be transmitted to the ground station for each motion planning decision, the control interval has to be large enough to accommodate the bandwidth and data transmission delay. The A* algorithm (one type of graph searching algorithms) is adopted to perform motion planning, and the detailed explanation of algorithm design and implementation can be found in our previous work [3].

Considering the distributed control architecture, a task allocation problem needs to be resolved to find the most suitable vehicle (e.g. with the shortest distance to the target) for each particular crowd group before finding the feasible cooperative trajectory. After one vehicle detects the location of a crowd group, the data is transmitted to the rest of the vehicles in the communication network. When other vehicles receive this information, they calculate their own distance to the location of the target and broadcast this computed information to all other vehicles in the system. Then, the one with the shortest distance will be selected to perform the motion planning based on its own prediction. In the cases of delay or data loss in the communication system, a timeout is set based on the speed of the crowd and communication latency. If this timeout is passed and no replies from other vehicles are received, the original vehicle, which has detected the target, will perform the motion planning to follow the crowd. The same graph-search algorithm - A* algorithm is used under the distributed control architecture. However, due to the lack of the control center, the following constraint (Equation (5)) has been considered for avoiding collisions among vehicles. Assume there exist \(l\) vehicles in the system, and \(D_v(t)\) represents the location of the selected vehicle for motion planning and \(D_i(t)\) represent another vehicles’ location at time \(t\). The constraint is defined as

$$\rho(D_v(t), D_i(t)) \geq \delta, \forall t \in \{1, 2, ..., T\}, i = 1, ..., v - 1, v + 1, ..., l$$ (5)

where, \(\rho\) is the distance function between two vehicles and \(\delta\) is defined as vehicle safety margin. This safety margin is based on the accuracy of vehicles’ localization system such as GPS sensor, which determines the geographical location of UAVs and UGVs. By employing the constraint, our algorithm (A*) will find a trajectory (i.e. a sequence of waypoints), where vehicles keep a minimum safety distance from all others at each time step.

As mentioned above, having information about the current vehicle position is crucial in their motion planning, but the problem is that the accuracy of GPS ranges up to 5 meters that may not be sufficient for motion planning. As a result, many research works have been done to resolve such problems, but most of them rely on the range data provided by the sensors (e.g. laser range finders, sonar, and stereo vision). In this work, computer vision was chosen for localization due to the following reasons. First of all, small scale UAVs and UGVs are not capable to lift heavy payloads such as laser range finders. Moreover, although the laser range finders fit well for the localization of ground vehicles, as the altitude of the car or mobile robot does not change relative to the course, they are not sufficient for estimating the position and orientation of an aerial vehicle. Last, richer information from a camera is needed to not only localize UAVs but also to detect humans and different objects from the scene.

![Figure 3: Illustration of visual odometry](image-url)
In order to do the localization, we need to 1) estimate the relative motion with respect to the previous frames, and 2) estimate the camera pose by landmark detection. In this paper we focus on the first step, which is called visual odometry. Figure 3 illustrates the overview of the visual odometry procedure, where the first stage is to extract keypoints from the images. In this stage, approaches for detecting features and extracting descriptor of the detected features need to be specified. There are numerous features that are widely used in the literature namely SIFT [17] and SURF features [18], Harris corners [13], FAST features [19], and oriented BRIEF features [20]. We chose FAST features for keypoint detection and BRIEF for keypoint descriptors after testing, because they are good, both in terms of accuracy and being applicable to the real time applications. Figures 4(a) and 4(b) illustrate a sample captured image and its detected features, respectively.

After keypoints extraction, features are matched between frame \( t \) and frame \( t - 1 \), where each feature is stored in a special data structure that keeps track of the features being matched in the previous frames (see Figure 4(c)). The transformation is estimated between every \( w \) frames due to the following reasons. First, larger motions can be estimated more accurately. Secondly, computational resources can be saved by doing so for every \( w \) frame. Lastly, inconsistent features can be eliminated because only features from frame \( t \) that has greater than or equal to \( w \) predecessors features are considered. As a result, this method turns out to be more efficient and robust. Then, the fundamental matrix \( F \) using 8-point algorithm and RANSAC are computed to remove the remaining outliers and the essential matrix \( E = K^T F K \), the 4 pairs of rotation, and translation from the essential matrix will be computed. In the final stage, the pair of rotation-translation involving most points with positive depth will be chosen. In the ideal case, the depth for all of the points should be positive, but some outlier points may pass through the RANSAC stage and the depth for those points become negative. Figure 4(d) shows such points in blue. It is worthy to note that the magnitude of transformation cannot be recovered using a single camera. As a result, the relative scale between consecutive translations needs to be estimated. It is assumed that the translation between frame \( t - 2w + 2 \) and frame \( t - w + 1 \) is known, and we want to find the scale of translation between frame \( t - w + 1 \) and \( t \). In order to do that, the distance of all the points, which are visible in all of the frames \( t - 2w + 2, t - w + 1, \) and \( t \) with respective to the position of the camera at time \( t - w + 1 \), should be estimated. Since the distance of the points grows linearly with the scale of translation, the relative scale is the median of the ratio of points’ distances between frames \( t - 2w + 2 \) and \( t - w + 1 \) over the distances of the same points between frame \( t - w + 1 \) and \( t \).
3. Integrated Control Testbed

3.1 Hardware (UAV/UGV)

The UAV platform chosen in this work is a custom-built quadrotor helicopter (see Figure 5(a)) consisting of off-the-shelf parts. The system is optimized for low cost and flexibility to carry various types of payloads, including a camera sensor and a powerful embedded computer to process computer vision. The aircraft frame is constructed from two G10/FR4 fiberglass-epoxy laminate plates, which mount four ABS plastic arms in a cross-like arrangement. At the end of each arm, a 22mm 1000Kv brushless DC outrunner motor is fixed in a vertical orientation. To each rotor, a 10-inch diameter, fixed pitch, carbon-fiber propeller is attached. Control of the aircraft is achieved by increasing or decreasing the speed of each rotor. Each rotor is connected to a small electronic circuit called an electronic speed controller (ESC), which is responsible for receiving speed commands from the flight computer and driving the phases of the brushless DC motors to vary the rotor speed. The flight microcontroller we have selected for the UAV platform is the ArduPilotMega (APM) 2.5, an open-source embedded autopilot platform designed by 3dRobotics. This autopilot is based on the Arduino Mega 2560 and features an integrated inertial measurement unit (IMU) that measures acceleration, rotational velocity, and the direction of the magnetic field in 3D. The APM processes the IMU data, serial commands, and user input from a 2.4GHz radio control signal, and sends speed commands to the ESCs for control and stabilization of the aircraft.

Two different UGV platforms have been built for our control testbed. The first UGV platform (see Figure 5(b)) is based on a small 1/16 scale toy remote control car platform, which has four wheels, and operates with car-like steering. It carries an APM to process IMU data, serial commands, and user input from a 2.4GHz radio and sends control signals to alter the speed and steering direction of the UGV. This UGV platform can travel at a relatively high speed, and also achieve rough control between waypoints via GPS. One of the limitations for the first UGV platform is the lacking of fine control of motion that is needed for operating the system within real environment. Therefore, a second UGV (see Figure 5(c)) has been designed in this work. This UGV embeds a roughly 11-inch-diameter differential drive platform with 9.5-inch polyurethane wheels connected to 29:1 DC gear motors with high resolution encoders. This platform was designed to travel at a top speed of 5 miles per hour, which is slightly faster than human walking speed.

![Assembled UAV](image1.png) ![Assembled four-wheel UGV](image2.png) ![Designed differential drive UGV](image3.png)

Figure 5: Developed hardware platforms

3.2 Hardware-in-the-loop Simulation

An agent-based hardware-in-the-loop simulation for our multi-vehicle system (see Figure 6(a)) has been developed for 1) centralized controller, and 2) mimicking behaviors of numerous UAVs/UGVs involving reduced hardware deployment costs (as many of them are simulated ones) without compromising performance. The integrated simulation model has been implemented in Repast Simphony (http://repast.sourceforge.net/repast_simphony.php) that also incorporates the terrain data from Geographical Information System (GIS) [21]. Furthermore, the algorithms (e.g. crowd detection, tracking, and motion planning) have been incorporated into the simulation package, which can interact with the real hardware through deployed hardware interface.
In addition, we have developed a physics-based simulation model (see Figure 6(b)) using Gazebo, which is an open source multi-robot simulation toolkit (http://gazebosim.org). This simulator can be used in Robotics Operating System (ROS), which provides a variety of plugins for integration with real UAVs and UGVs. In our implementation, hector_quadrotor package that was developed by a research group at Technische Universität Darmstadt (http://wiki.ros.org/hector_quadrotor) was employed. This package provides a quadrotor model for controlling in Gazebo simulator.

4. Experiments

In surveillance and crowd control missions, computational latency, which is defined as average processing time of the embedded algorithms, is a key factor that influences the system performance and control interval of unmanned vehicles. Although the walking speed of individuals in the crowd is normally less than those of the vehicles, improving the computational latency can make the system more flexible to adapt to changes (e.g. environmental disturbance, obstacles). In the rest of this section, several experiment results conducted in our integrated control testbed are presented to compare the computational latency of our proposed algorithms.

In the experiments, we focus on two widely adopted architectures (i.e. centralized and distributed). For the centralized architecture, a 2.8 GHz quad-core Intel Pentium x86 that runs Ubuntu Linux was used as the processing unit in the control center. For the distributed architecture, several options were evaluated with the purpose to find an inexpensive platform that is lightweight and offers sufficient computing power for UAV/UGV. A few compact onboard computer options include the popular ARM Cortex-A8 based Gumstix Overo that is extremely small and lightweight (about the size of a stick of gum), and the credit-card-sized Raspberry Pi that is inexpensive and widely adopted by the community. For the UAV and UGV testbed platforms, we selected the ODROID-U2/3, which is a 1.7GHz quad-core ARM-Cortex-A9 single-board computer (SBC) that runs Ubuntu Linux. This platform offers significantly more computing power than the Gumstix or Raspberry Pi within an extremely small package.

Table 1 summarizes the computational latency of each algorithm under the centralized and distributed architectures. Average processing time per frame in crowd detection and visual odometry of motion planning are compared inside the table. As shown in the table, the processing time per frame under the distributed control architecture is roughly 3–5 times longer than that under the centralized control architecture. Now considering the tracking case, information sharing is known as an important factor in the real implementation of the proposed information-aggregation-based tracking algorithm. Under the centralized control architecture, the information sharing doesn’t have impact, as all the data will be transmitted to the control center for processing. However, under the distributed control architecture, two extreme cases are considered before the tracking algorithm performs: 1) complete information sharing of crowd detection data to all the UAVs and UGVs, and 2) no information sharing of crowd detection data among UAVs/UGVs. The crowd coverage rate evaluates the accuracy of the tracking algorithm, which can be computed as the ratio of the estimated crowd region over the actual crowd region. Our preliminary results showed that the tracking algorithm with no information sharing is much less accurate than tracking with complete information sharing (0.65 vs. 0.95). Based on these results, under the centralized control architecture our system can perform with minimum control interval of 0.097 seconds. Under the distributed architecture, the control interval can be selected as low as 0.506 seconds using parallel processing. In case of no parallel processing capability, control interval should be defined based on the total processing time of all the algorithms. It is worthy to note that the
control interval can be influenced by other constraints (e.g. communicational latency) in the communication system, which is discussed in the next two paragraphs.

Table 1: Computational latency of implemented algorithms under different control architectures

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Centralized Architecture</th>
<th>Distributed Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crowd Detection</td>
<td>0.097 seconds per frame</td>
<td>0.506 seconds per frame</td>
</tr>
<tr>
<td>Crowd Tracking with information sharing</td>
<td>0.08 seconds</td>
<td>0.25 seconds</td>
</tr>
<tr>
<td>Crowd Tracking without information sharing</td>
<td></td>
<td>0.16 seconds</td>
</tr>
<tr>
<td>Motion Planning - Visual Odometry</td>
<td>0.05 seconds per frame</td>
<td>0.22 seconds per frame</td>
</tr>
</tbody>
</table>

In the centralized control architecture, it is important for the vehicles to send all of their sensory information to the central controller. While the data rate offered by the XBee PRO 900HP is sufficient for most types of data, it is not sufficient for live transmission of high definition video data that can be several megabits per second. In addition, while Wi-Fi offers a significantly higher data throughput rate than XBee, it is limited in range to only 100m. We are currently investigating ways to improve the communication range of our centralized infrastructure. For example, it may be possible to extend the range of the wireless network by improving the wireless network infrastructure (i.e. adding more access points), or leveraging cellular data infrastructure (i.e. using a 3G/4G cell phone network). It is also possible to transmit video via a more dedicated means such as consumer video transmitters (e.g. first-person-view hobby video transmitters, baby monitors, wireless CCTV), which offer low-resolution analog video at a moderate range of 1-2km.

In the distributed control architecture, a communication network among the UAVs and UGVs is critical for the vehicles to broadcast their own status and monitor the status of other vehicles. For simultaneous communication between the ground station, multiple UAVs, and UGVs, two options have been evaluated: Wi-Fi (IEEE 802.11.x standard over 2.4GHz) and XBee PRO 900HP (FHSS over 900MHz). For the outdoor application, the XBee PRO 900HP is preferred because it offers a long-range communication (up to 45km) and mesh networking capability, with a maximum data rate of 200kbps.

5. Conclusions and Future Works
In this work, a comparative analysis of different control architectures (e.g. centralized vs. distributed) in multi-vehicle systems was conducted to demonstrate their applicability for collaborative surveillance and crowd control. Crowd detection, tracking and motion planning algorithms were developed or refined, and implemented under different control architectures. Furthermore, an integrated control testbed consisting of real hardware (UAVs and UGVs) and software (agent-based and physics-based simulations) was developed. Using this testbed, experiments were conducted by the proposed algorithms based on different control architectures to measure their computational latency. Preliminary results showed that in the distributed control architecture, with no information sharing among the vehicles, the computational latency and the control interval could be reduced significantly. However, the system performance could be impacted under this circumstance as well. As discussed in the experiments section, in addition to the computational latency, the communicational latency has also a high impact on the system performance and control intervals. Future studies will be conducted on different communication systems by transmitting and receiving required information in different architectures and testing their actual latency. Furthermore, additional experiments using physics-based simulations would allow us to perform more accurate evaluation of the overall system performance under different control architectures in a hybrid environment involving real vehicles and several simulated vehicles (based on detailed physics).

Acknowledgments
This work is financially supported by the Air Force Office of Scientific Research under FA9550-12-1-0238 (A part of Dynamic Data Driven Application Systems (DDDAS) projects).
References


